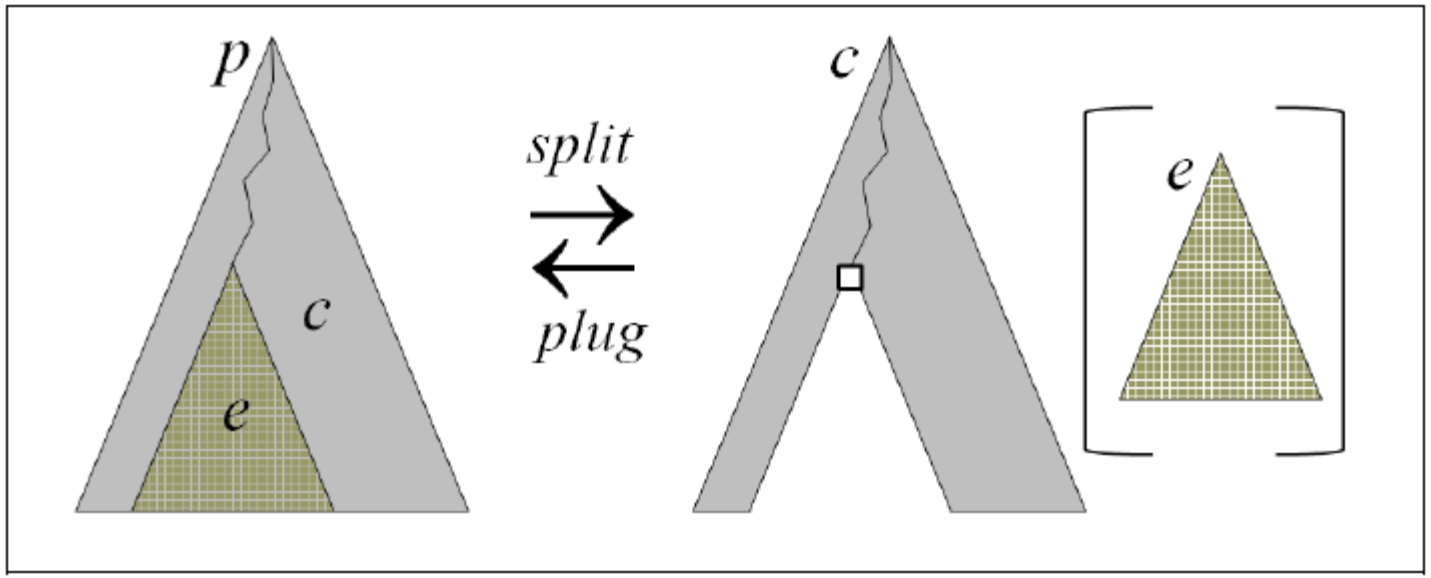
Reduction Semantics with Evaluation Contexts (RSEC)

- Matthias Felleisen and collaborators (1992)
- Previous approaches encoded the program execution context as a proof context, by means of rule conditions or premises
  - This has a series of advantages, but makes it hard to define control intensive features, such as abrupt termination, exceptions, call/cc, etc.
- We would like to have the execution context explicit, so that we can easily save it, change it, or even delete it
- Reduction semantics with evaluation contexts does precisely that
  - It allows to formally define *evaluation contexts*
  - Rules become mostly *unconditional*
  - Reductions can only happen "*in context*"

Splitting and Plugging

- RSEC relies on reversible implicit mechanisms to
  - Split syntax into an evaluation context and a redex
  - Plug a redex into an evaluation contexts and obtain syntax again



$$p = c[e]$$

Evaluation Contexts

- Evaluation contexts are typically defined by the same means that we use to define the language syntax, that is, grammars
- The *hole* □ represents the place where redex is to be plugged
- Example:

$$
\begin{array}{rcl}
\mathit{Context} & ::= & \square \\
& | & \mathit{Context} \texttt{ <= } \mathit{AExp} \\
& | & \mathit{Int} \texttt{ <= } \mathit{Context} \\
& | & \mathit{Id} \texttt{ := } \mathit{Context} \\
& | & \mathit{Context} \texttt{ ; } \mathit{Stmt} \\
& | & \texttt{if } \mathit{Context} \texttt{ then } \mathit{Stmt} \texttt{ else } \mathit{Stmt} \\
& | & \ldots
\end{array}
$$

Correct Evaluation Contexts

$\square$

$3 \texttt{ <= } \square$

$\square \texttt{ <= } 3$

$\square \texttt{ ; } x \texttt{ := } 5$

$\texttt{if } \square \texttt{ then } s_1 \texttt{ else } s_2$

Wrong Evaluation Contexts

$\square$ <= $\square$

x <= 3

x <= $\square$

x := 5 ; $\square$

$\square$:= 5

if x <= 7 then $\square$ else x := 5

Splitting/Plugging of Syntax

$$7 \;=\; (\Box)[7]$$

$$3 <= x \;=\; (3 <= \Box)[x] \;=\; (\Box <= x)[3] \;=\; (\Box)[3 <= x]$$

$$3 <= (2 + x) + 7 \;=\; (3 <= \Box + 7)[2 + x] \;=\; (\Box <= (2 + x) + 7)[3] \;=\; \ldots$$

Characteristic Rule of RSEC

$$\frac{e \to e'}{c[e] \to c[e']}$$

- The characteristic rule of RSEC allows us to only define semantic rules stating how redexes are reduced
  - This significantly reduces the number of rules
  - The semantic rules are mostly unconditional (no premises)
  - The overall result is a semantics which is compact and easy to read and understand

RSEC of IMP – Evaluation Contexts

IMP evaluation contexts syntax	IMP language syntax
$Context$ ::= $\square$ | $Context$ + $AExp$ | $AExp$ + $Context$ | $Context$ / $AExp$ | $AExp$ / $Context$ | $Context$ <= $AExp$ | Int <= Cxt | not $Context$ | $Context$ and $BExp$ | Id := $Context$ | $Context$; $Stmt$ | if $Context$ then $Stmt$ else $Stmt$	$AExp$ ::= Int | Id | | $AExp$ + $AExp$ | $AExp$ / $AExp$ $BExp$ ::= $Bool$ | $AExp$ <= $AExp$ | not $BExp$ | $BExp$ and $BExp$ $Stmt$ ::= Id := $AExp$ | $Stmt$; $Stmt$ | if $BExp$ then $Stmt$ else $Stmt$ | while $BExp$ do $Stmt$ Pgm ::= vars **List**{Id} ; $Stmt$

RSEC of IMP – Rules

$$Context ::= \ldots \mid \langle Context, State \rangle$$

$$\frac{e \to e'}{c[e] \to c[e']}$$

$$\begin{aligned}
\langle c, \sigma \rangle[x] &\to \langle c, \sigma \rangle[\sigma(x)] \\
i_1 \texttt{+} i_2 &\to i_1 +_{Int} i_2 \\
i_1 \texttt{/} i_2 &\to i_1 /_{Int} i_2, \quad \text{when } i_2 \neq 0 \\
i_1 \texttt{<=} i_2 &\to i_1 \leq_{Int} i_2 \\
\texttt{not true} &\to \texttt{false} \\
\texttt{not false} &\to \texttt{true} \\
\texttt{true and}\, b_2 &\to b_2 \\
\texttt{false and}\, b_2 &\to \texttt{false} \\
\langle c, \sigma \rangle[x \texttt{:=} i] &\to \langle c, \sigma[i/x] \rangle[\texttt{skip}] \\
\texttt{skip ;}\, s_2 &\to s_2 \\
\texttt{if true then}\, s_1 \,\texttt{else}\, s_2 &\to s_1 \\
\texttt{if false then}\, s_1 \,\texttt{else}\, s_2 &\to s_2 \\
\texttt{while}\, b \,\texttt{do}\, s &\to \texttt{if}\, b \,\texttt{then}\, (s \texttt{ ; while}\, b \,\texttt{do}\, s) \,\texttt{else skip} \\
\langle \texttt{vars}\, xl \texttt{ ;}\, s \rangle &\to \langle s, (xl \mapsto 0) \rangle
\end{aligned}$$

RSEC Derivation

$$\langle \boxed{x := 1}\ ;\ y := 2\ ;\ \texttt{if}\ x <= y\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 0, y \mapsto 0)\rangle$$

$$\rightarrow\ \langle \boxed{\texttt{skip}\ ;\ y := 2}\ ;\ \texttt{if}\ x <= y\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 0)\rangle$$

$$\rightarrow\ \langle \boxed{y := 2}\ ;\ \texttt{if}\ x <= y\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 0)\rangle$$

$$\rightarrow\ \langle \texttt{skip}\ ;\ \texttt{if}\ x <= y\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle \texttt{if}\ \boxed{x}\ <= y\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle \texttt{if}\ 1 <= \boxed{y}\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle \texttt{if}\ \boxed{1 <= 2}\ \texttt{then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle \texttt{if true then}\ x := 0\ \texttt{else}\ y := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle x := 0\ ,\ (x \mapsto 1, y \mapsto 2)\rangle$$

$$\rightarrow\ \langle \texttt{skip}\ ,\ (x \mapsto 0, y \mapsto 2)\rangle$$

RSEC in Rewriting Logic

- Like with the other styles, RSEC can also be faithfully represented in rewriting logic and, implicitly, in Maude
- However, RSEC is context sensitive, while rewriting logic is not (rewriting logic allows rewriting strategies, but one can still not match and modify the context, as we can do in RSEC)
- We therefore need
  - A mechanism to achieve context sensitivity (the splitting/plugging) in rewriting logic
  - Use that mechanism to represent the characteristic rule of RSEC

Evaluation Contexts in Rewriting Logic

- An evaluation context CFG production in RSEC has the form

$$Context \quad ::= \quad \pi(N_1, \ldots, N_n, Context)$$

- Associate to each such production one rule and one equation:

$$split(\pi(T_1, \ldots, T_n, T)) \rightarrow \pi(T_1, \ldots, T_n, C)[Syn] \ \textbf{if} \ split(T) \rightarrow C[Syn]$$

$$plug(\pi(T_1, \ldots, T_n, C)[Syn]) = \pi(T_1, \ldots, T_n, plug(C[Syn]))$$

- Plus, we add one generic rule and one generic equation:

$$split(Syn) \rightarrow \Box[Syn]$$
$$plug(\Box[Syn]) = Syn$$

IMP Examples:

- For productions

$$\begin{array}{rcl} Context & ::= & Context \texttt{ <= } AExp \\ & | & Int \texttt{ <= } Context \\ & | & Id \texttt{ := } Context \end{array}$$

we add the following rewrite logic rules and equations:

$$split(A_1 \texttt{ <= } A_2) \rightarrow (C \texttt{ <= } A_2)[Syn] \textbf{ if } split(A_1) \rightarrow C[Syn]$$
$$plug((C \texttt{ <= } A_2)[Syn]) = plug(C[Syn]) \texttt{ <= } A_2$$

$$split(I_1 \texttt{ <= } A_2) \rightarrow (I_1 \texttt{ <= } C)[Syn] \textbf{ if } split(A_2) \rightarrow C[Syn]$$
$$plug((I_1 \texttt{ <= } C)[Syn]) = I_1 \texttt{ <= } plug(C[Syn])$$

$$split(X \texttt{ := } A) \rightarrow (X \texttt{ := } C)[Syn] \textbf{ if } split(A) \rightarrow C[Syn]$$
$$plug((X \texttt{ := } C)[Syn]) = X \texttt{ := } plug(C[Syn])$$

RSEC Reduction Rules in Rewriting Logic

// for each term l that appears as the left-hand side of a reduction rule
// "$l(c_1[l_1], \ldots, c_n[l_n]) \rightarrow \ldots$", add the following conditional
// rewrite rule (there could be one l for many reduction rules):

$$\circ\overline{l}(T_1, \ldots, T_n) \rightarrow T \ \textbf{if} \ plug(\circ\overline{l}(split(T_1), \ldots, split(T_n))) \rightarrow T$$

// for each non-identity term r appearing as right-hand side in a reduction rule
// "$\ldots \rightarrow r(c_1[r_1], \ldots, c_n[r_n])$", add the following equation
// (there could be one r for many reduction rules):

$$plug(\overline{r}(Syn_1, \ldots, Syn_n)) = \overline{r}(plug(Syn_1), \ldots, plug(Syn_n))$$

// for each reduction semantics rule "$l(c_1[l_1], \ldots, c_n[l_n]) \rightarrow r(c'_1[r_1], \ldots, c'_{n'}[r_{n'}])$"
// add the following "semantic" rewrite rule:

$$\circ\overline{l}(\overline{c_1}[\overline{l_1}], \ldots, \overline{c_n}[\overline{l_n}]) \rightarrow \overline{r}(\overline{c'_1}[\overline{r_1}], \ldots, \overline{c'_n}[\overline{r_{n'}}])$$

IMP Examples

- One rule of first kind

$$\circ\, Cfg \rightarrow Cfg' \ \textbf{if}\ plug(\circ\, split(Cfg)) \rightarrow Cfg'$$

- No need for equations of second kind
- Characteristic rule of RSEC:

$$\circ\, C[Syn] \rightarrow C[Syn'] \ \textbf{if}\ C \neq \Box \wedge \circ\, Syn \rightarrow Syn'$$

- The remaining rules are as natural as can be:

$$\circ\, I_1 \texttt{<=} I_2 \rightarrow I_1 \leq_{Int} I_2$$
$$\circ\, \texttt{skip ;}\ S_2 \rightarrow S_2$$
$$\circ\, \texttt{if true then}\ S_1\ \texttt{else}\ S_2 \rightarrow S_1$$